Creación de Aplicaciones Python

Lección 9: API REST con framework FastAPI

eip ESCUELA INTERNACIONAL POSGRADOS

ÍNDICE

Lección 9: API REST con framework FastAPI

PRESENTACIÓN Y OBJETIVOS

En esta lección aprenderemos a realizar un API REST usando FastAPI como Framework de Python, realizando los métodos HTTP más importante como son GET, POST, PUT y DELETE.

Objetivos

- *Conocer el framework de Python FastAPI*
- *Realizar una API REST empleando FastAPI.*

1. ¿ Qué es FastAPI ? ¿ Por qué usarlo?

FastAPI es un web framework moderno y rápido (de alto rendimiento) para construir APIs con Python 3.6+ basado en Python.

Sus características principales son:

1) Rapidez: Alto rendimiento, a la par con NodeJS y Go (gracias a Starlette y Pydantic). Uno de los frameworks de Python más rápidos.
2) Rápido de programar: Incrementa la velocidad de desarrollo entre 200% y 300%.
3) Menos errores: Reduce los errores humanos (de programador) aproximadamente un 40%.
4) Intuitivo: Gran soporte en los editores con auto completado en todas partes.
5) Fácil: Está diseñado para ser fácil de usar y aprender. Gastando menos tiempo leyendo documentación.
6) Corto: Minimiza la duplicación de código. Múltiples funcionalidades con cada declaración de parámetros. Menos errores.
7) Robusto: Crea código listo para producción con documentación automática interactiva.
8) Basado en estándares: Basado y totalmente compatible con los estándares abiertos para APIs: OpenAPI (conocido previamente como Swagger) y JSON Schema.

2. CREANDO UNA API REST CON FASTAPI

Creamos una carpeta en la carpeta de proyectos de Atom llamada FastAPI y la añadimos al IDE de Atom:

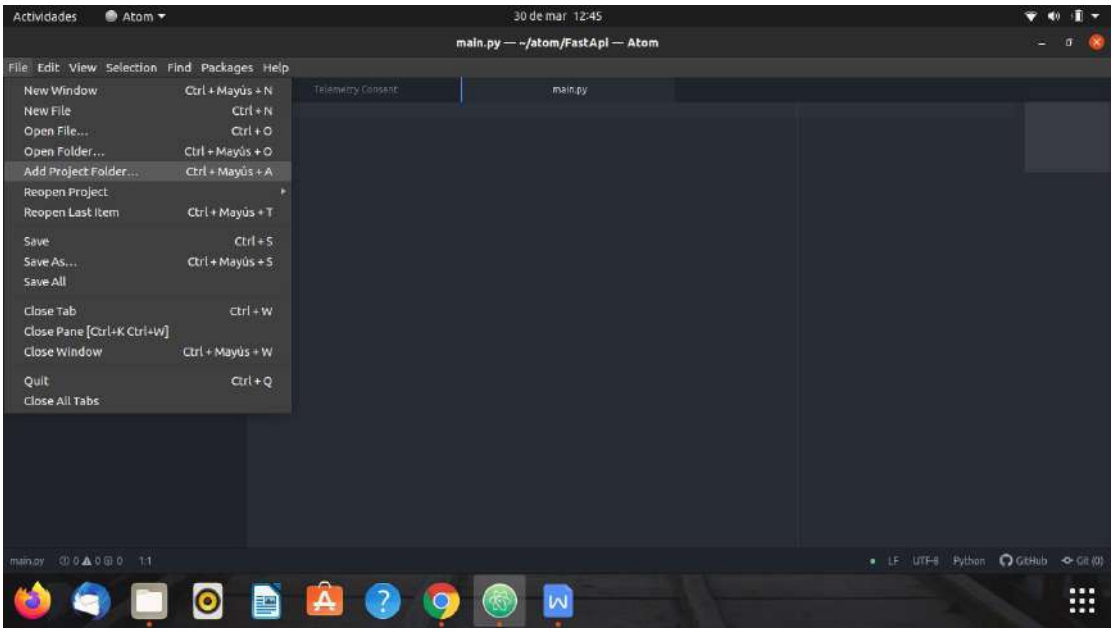

Figura 2.1. Añadir el proyecto a IDE Atom

Nos abre las carpetas y seleccionamos la carpeta que acabamos de crear FastAPI y damos aceptar:

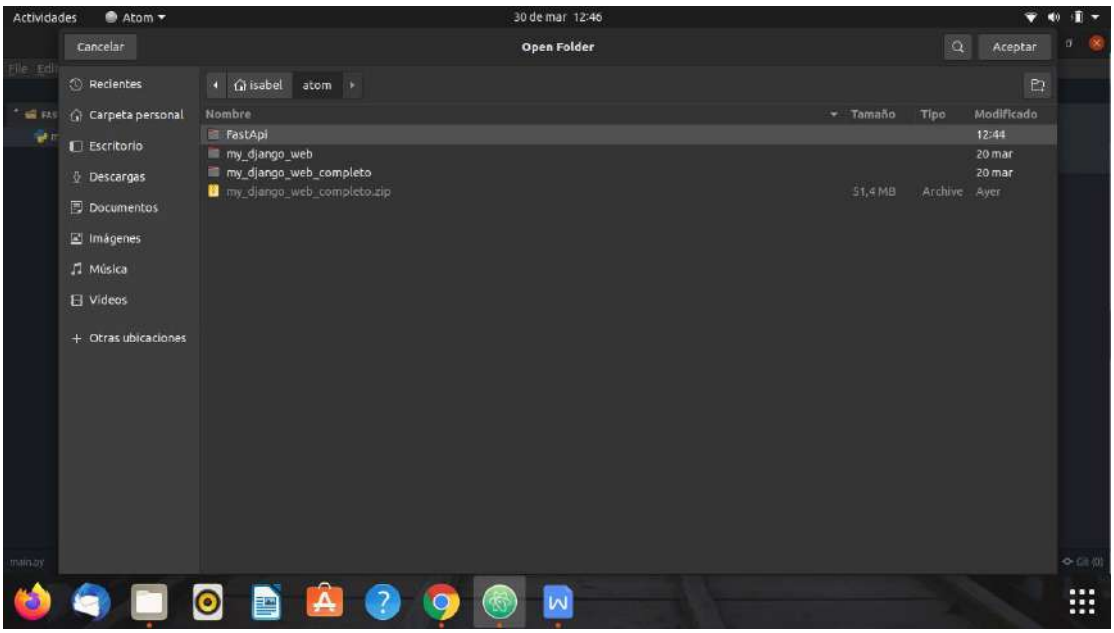

Figura 2.2. Añadir el proyecto a IDE Atom

Nos aparece la carpeta de nuestro proyecto en el entorno de Atom:

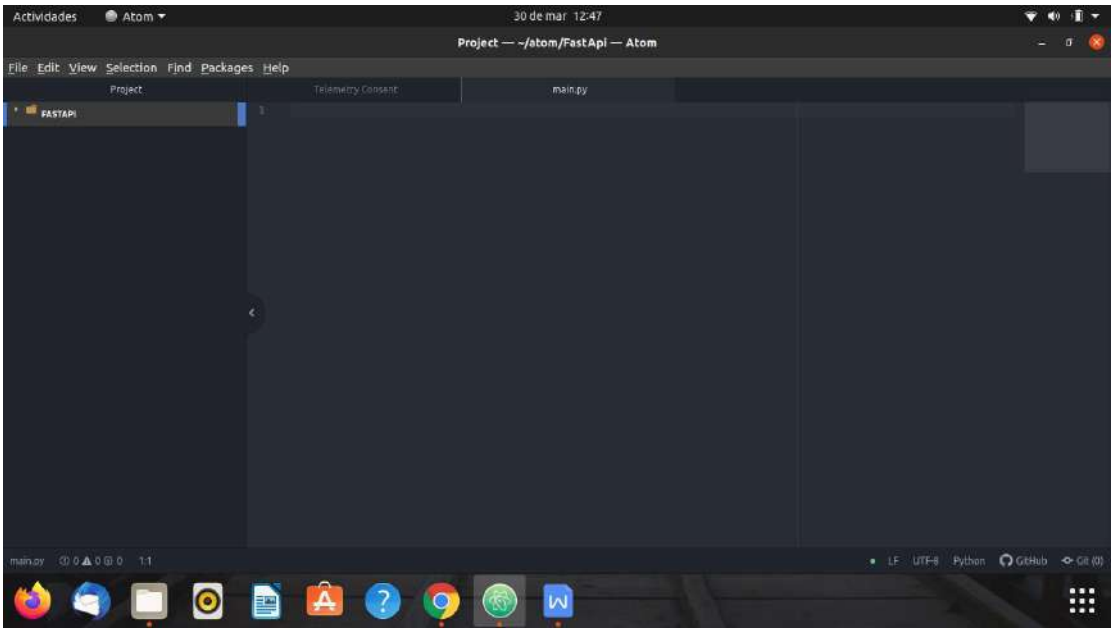

Figura 2.3. Añadir el proyecto a IDE Atom

Ahora creamos un archivo dentro de esa carpeta para ello pulsamos con el botón derecho sobre la carpeta y seleccionamos New File:

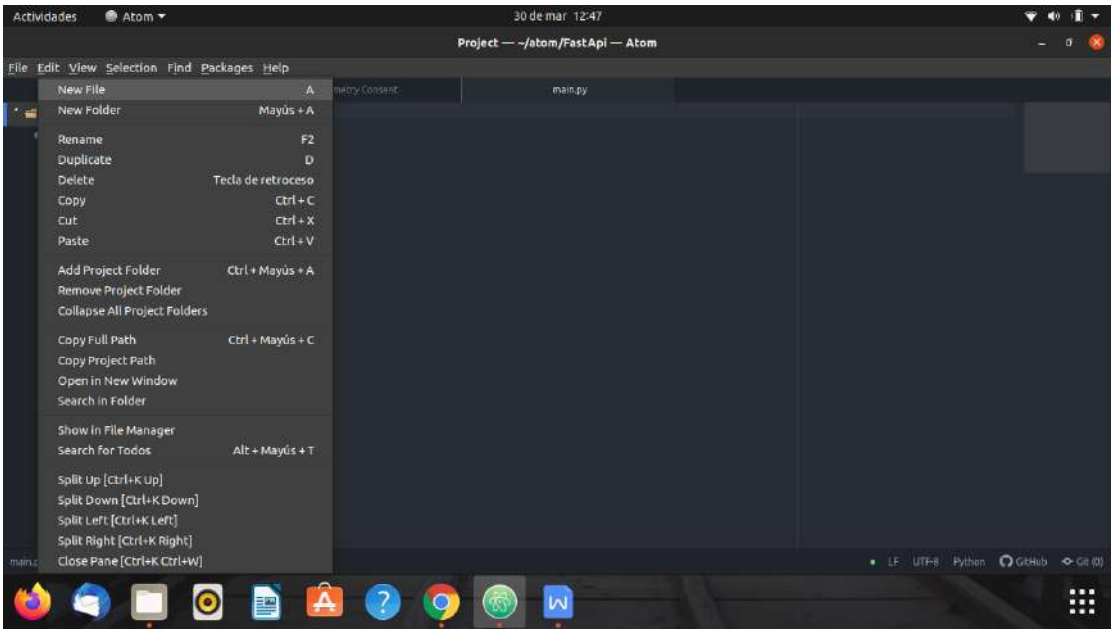

Figura 2.4. Crear archivo main.py donde escribiremos nuestro código en el IDE Atom

Le llamamos main.py y lo creamos:

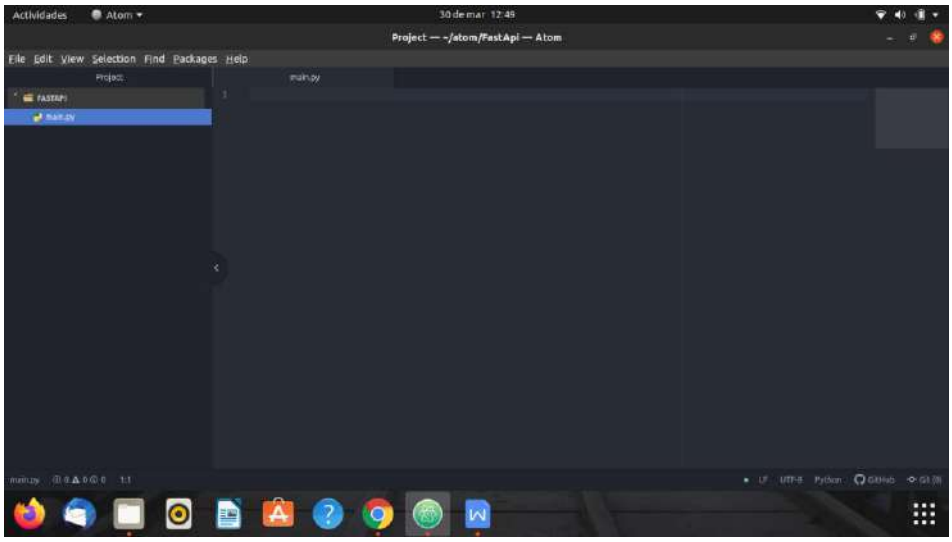

Figura 2.5. Crear archivo main.py donde escribiremos nuestro código en el IDE Atom

Escribiremos el siguiente código para crear nuestra primera aplicación usando FastAPI:

```
from fastapi import FastAPI

# creamos la primera aplicación fastAPI llamada “app”:

app = FastAPI()

# Método GET a la url "/"

# llamaremos a nuestra aplicación (<app name> + <método permitido>)

@app.get("/")

async def root():

    # Retornar el mensaje bienvenido a FastAPI

    return {"message": "Welcome to FastAPI!!"}
```

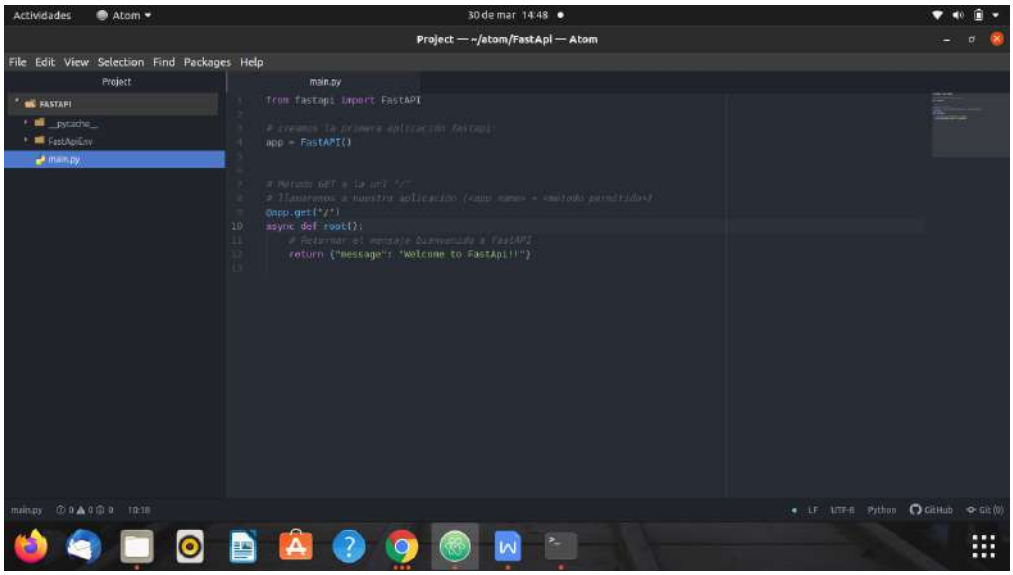

Figura 2.6. Archivo main.py se observa el código en Python de la aplicación FastAPI

Una vez hecho vamos al terminal donde tenemos nuestro script y creamos el entorno virtual para trabajar con esta aplicación:

```
virtualenv FastApiEnv
```

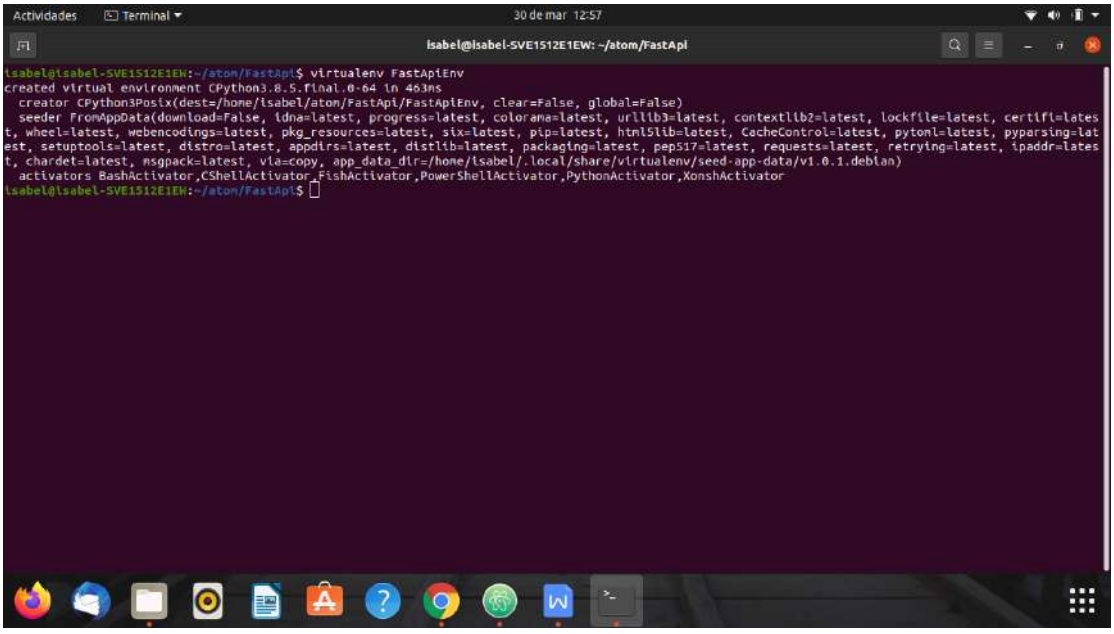

Figura 2.7. Entorno Virtual para trabajar con el proyecto FastAPI

Entramos en el Entorno Virtual:

```
source FastApiEnv/bin/activate
```

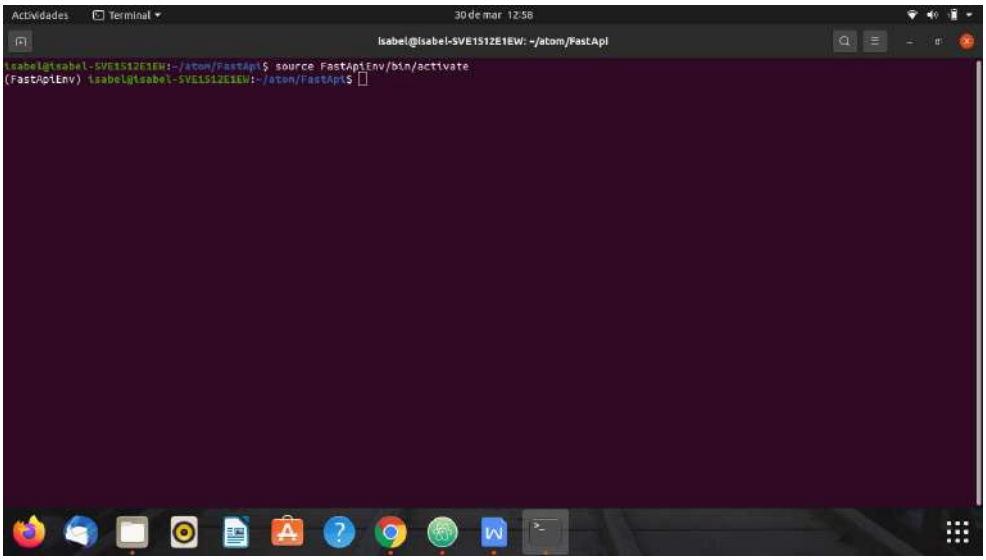

Figura 2.8. Activar el Entorno Virtual para trabajar con el proyecto FastAPI

Instalamos la librería fastapi y uvicorn:

```
pip install fastapi

pip install uvicorn[standard]
```

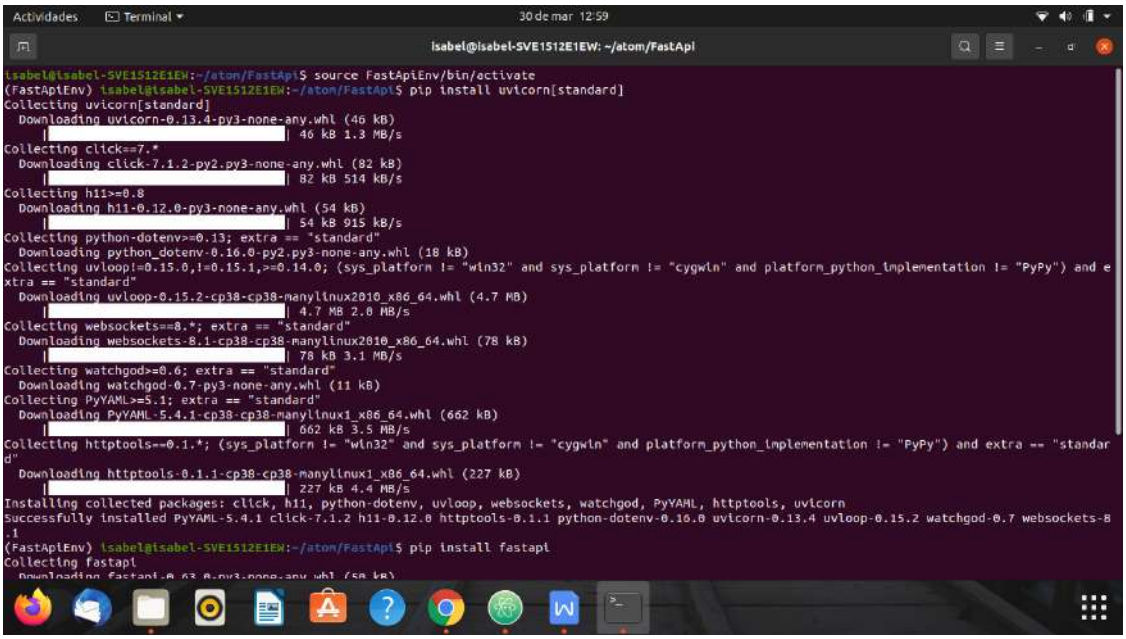

Figura 2.9. Instalación de uvicon[standard]

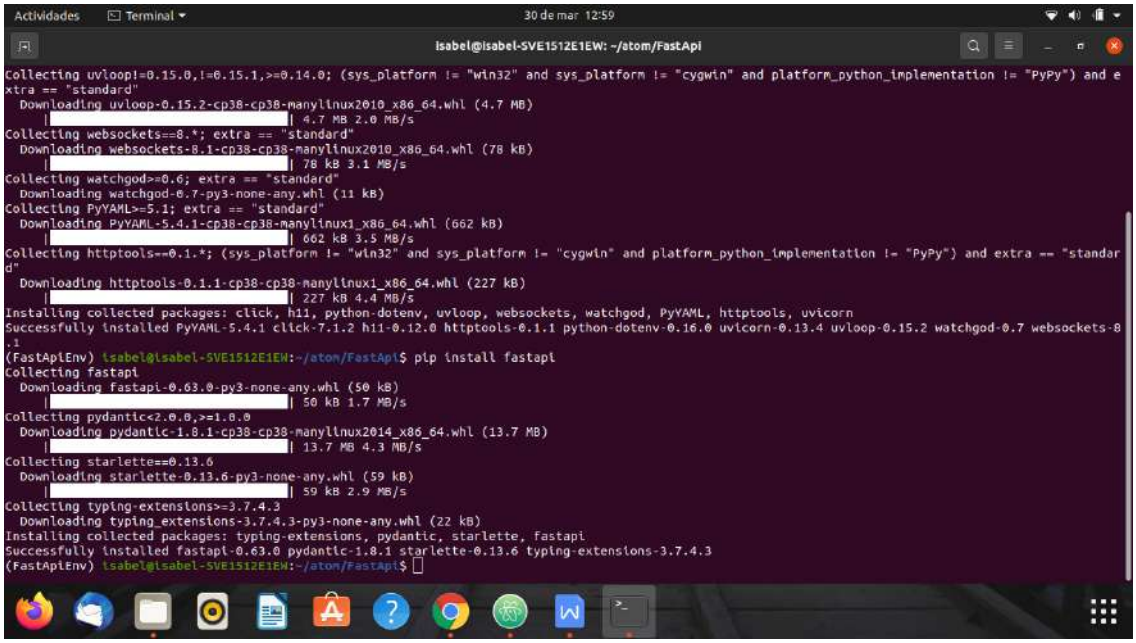

Figura 2.10. Instalación de FastAPI

Para ejecutar esta aplicación debemos de poner:

```
uvicorn main:app --reload
```

El comando uvicorn main:app se refiere a:

- main: el archivo main.py (“módulo” de python)
- app: el objeto creado dentro de main.py en la línea app = FastAPI()
- --reload: hace que la aplicación se reinicie después de un cambio de código. Sólo se usa en desarrollo.

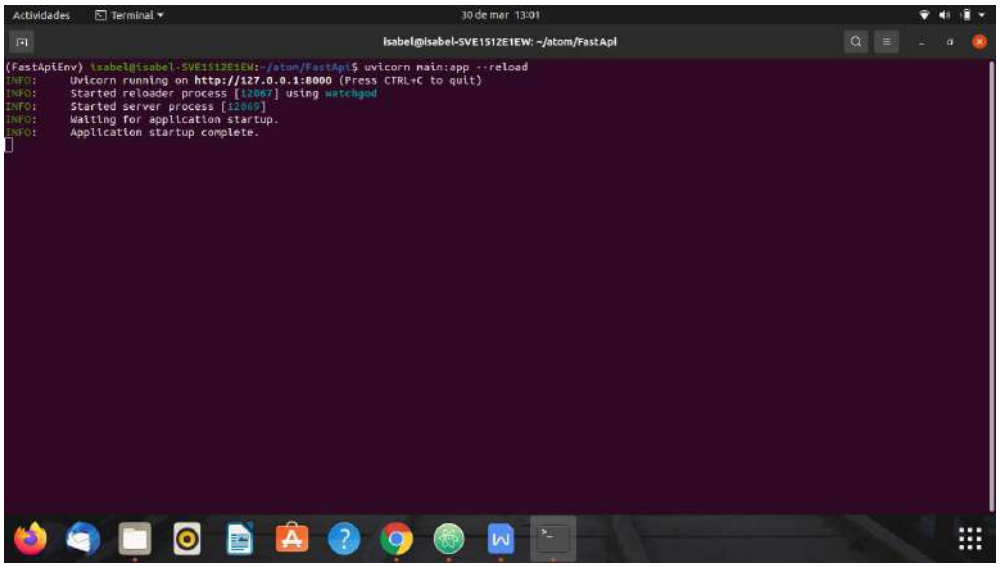

Figura 2.11. Ejecutar la aplicación FastAPI

Si vamos al navegador y ponemos: http://127.0.0.1:8000/

Observamos que nos muestra el mensaje de bienvenido a FastAPI

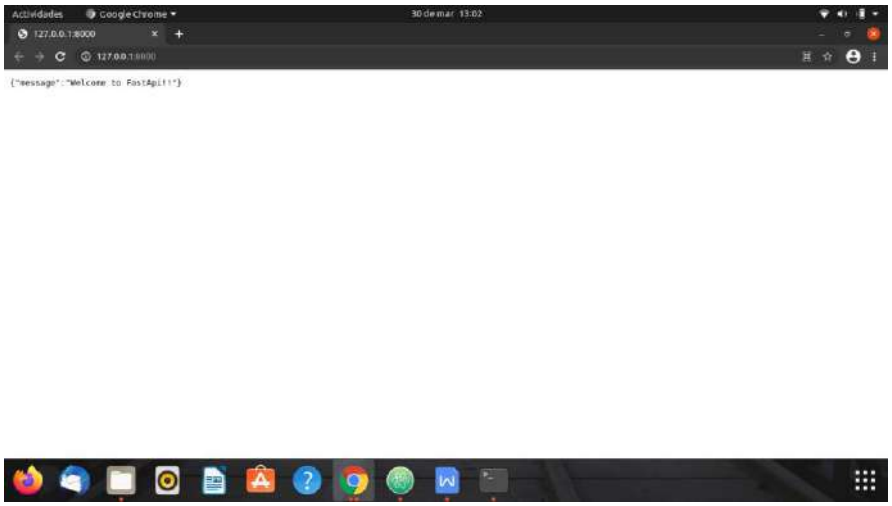

Figura 2.12. Ejemplo de cómo se ve la aplicación al ejecutarla en el navegador

Si ponemos: http://127.0.0.1:8000/docs

Se observa los métodos permitidos por la aplicación que hemos programado (GET).

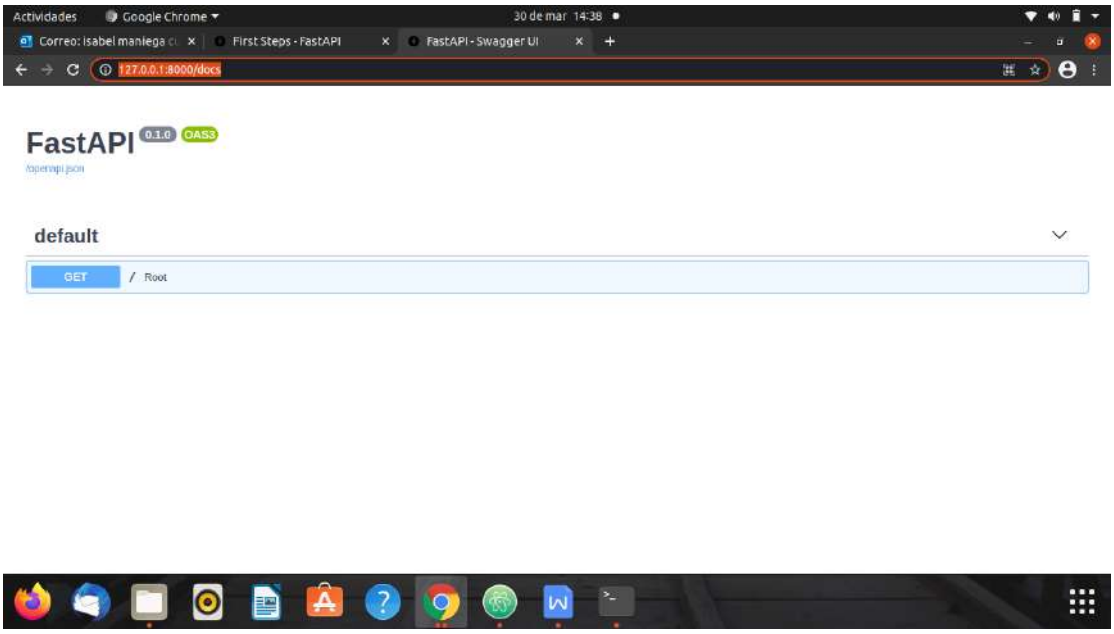

Figura 2.13. Ejemplo de la uri "/docs" al ejecutarla en el navegador

Esto nos evitará tener que usar otro tipo de herramientas como POSTMAN.

Si desplegamos el menú de GET se observa un botón de ***Try it out*** nos permite ejecutar el método GET, una sección de ***Parameters*** para enviar los parámetros y ***Responses*** donde nos mostrará la respuesta.

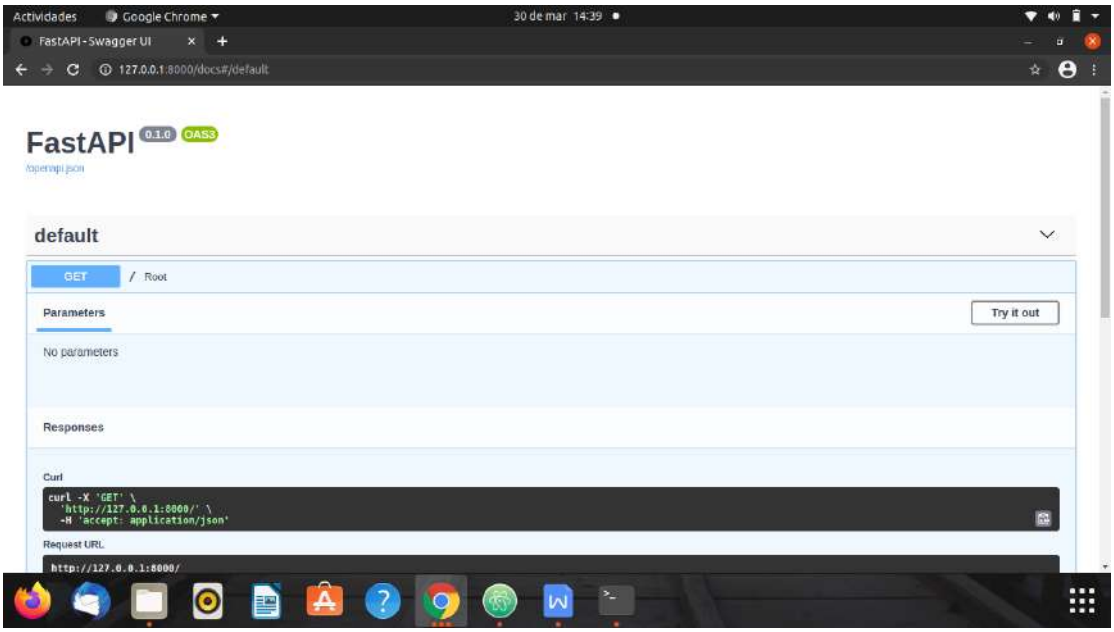

Figura 2.14. Ejemplo del método GET

Hacemos clic en ***Try it out***:

Nos despliega dos botones ***Execute*** y ***Clear***:

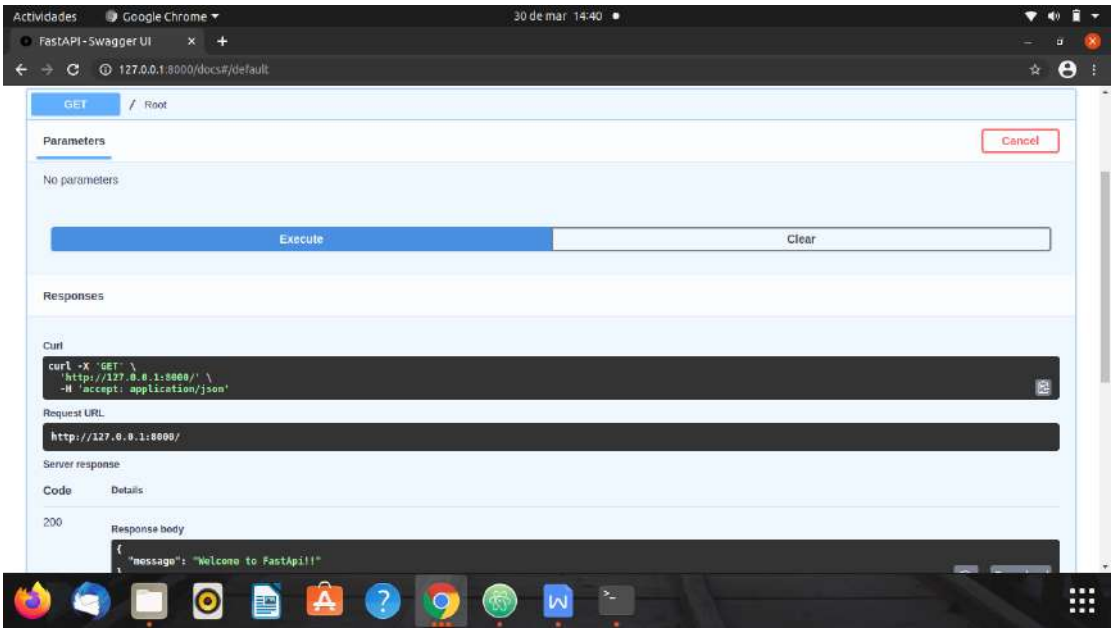

Figura 2.15. Ejemplo del método GET pulsando "Execute" se observa como respuesta 200 OK y el mensaje Bienvenido a FastAPI

Si damos a ***Execute*** vemos que debajo en ***Response*** nos muestra un 200 Ok y la respuesta que es nuestro mensaje:

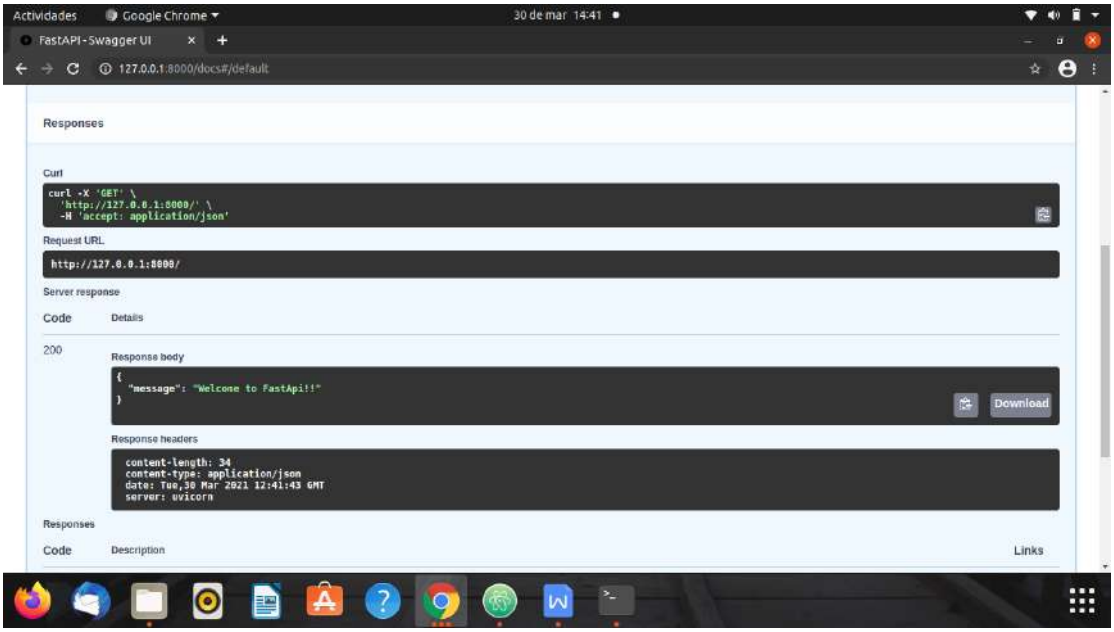

Figura 2.16. Ejemplo del método GET pulsando "Execute" se observa como respuesta 200 OK y el mensaje Bienvenido a FastAPI

3. API REST - IRIS DATASET

3.1. Método GET

Vamos a emplear el mismo ejemplo que usamos para Django mediante el CSV del conjunto de datos de Iris Dataset.

Lo primero será necesario instalar en nuestro entorno la librería pandas:

```
pip install pandas
```

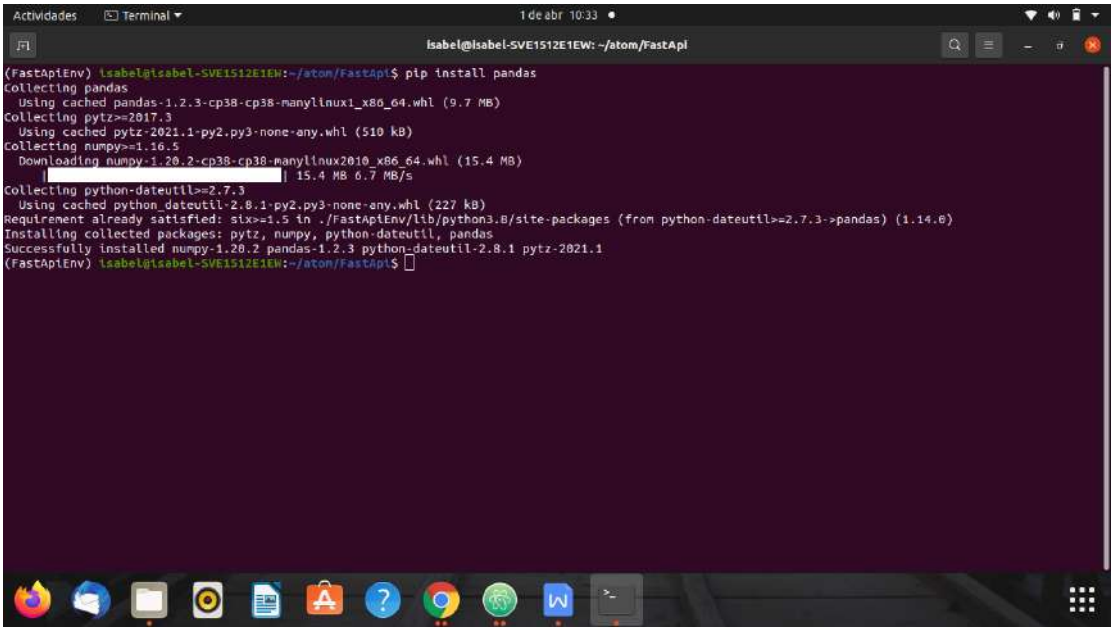

Figura 3.1. Instalación de la librería pandas

A continuación en nuestro archivo main.py importamos las siguientes librerías:

```
import pandas as pd --> pandas para lectura en formato dataframe

import json --> json para transformarlo en formato json

import csv --> csv para insertar los datos en él

import os --> para reconocer el entorno y usarlo como consola


MEDIA_ROOT = os.path.expanduser("~/atom/FASTAPI/iris.csv") --> ruta donde se
encuentra nuestro archivo.

# Método GET a la url "/iris/"

# llamaremos a nuestra aplicación (<app name> + <método permitido>)
```

```
@app.get("/iris/")
async def root():
    # Cargamos el dataset con ayuda de pandas:
    X_df = pd.read_csv(MEDIA_ROOT)
    # Lo transformamos a json:
    data = X_df.to_json(orient="index")
    data = json.loads(data)
    # Retornar el dataset
    return data
```

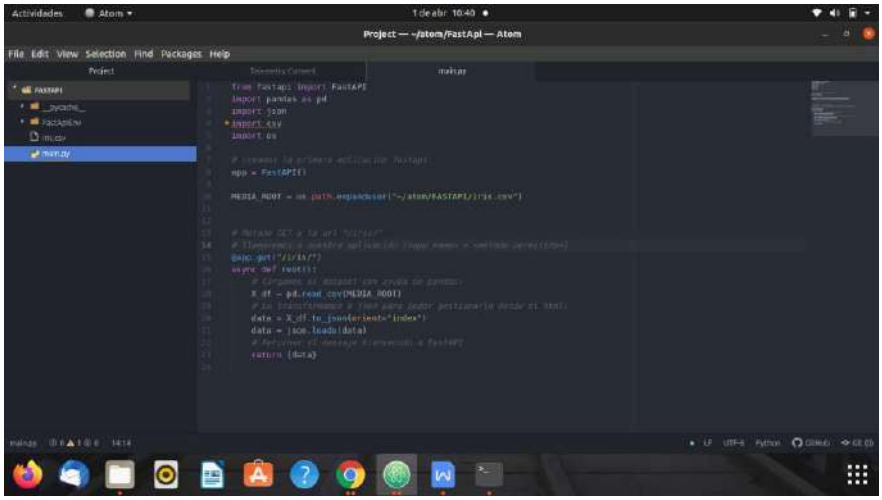

Figura 3.2. Ejemplo de programación del método GET en FastAPI con Iris Dataset

Ejecutamos la aplicación:

```
uvicorn main:app --reload
```

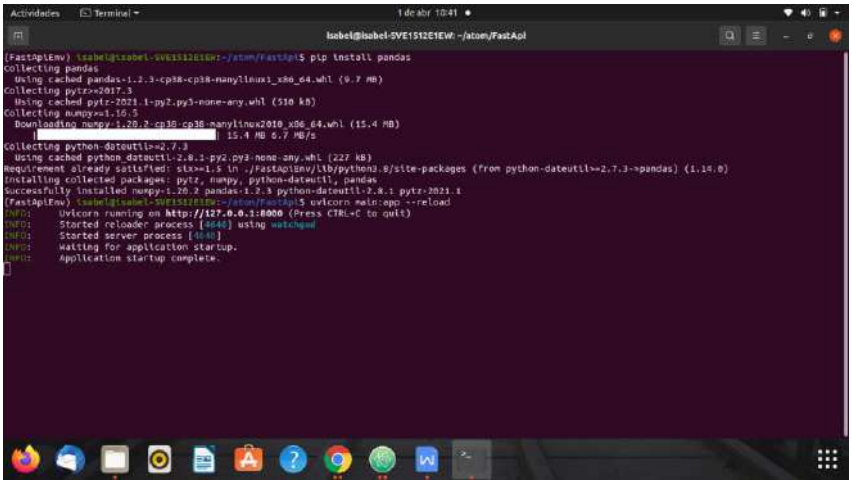

Figura 3.3. Ejecución de FastAPI

Vamos a la página http://127.0.0.1:8000/docs

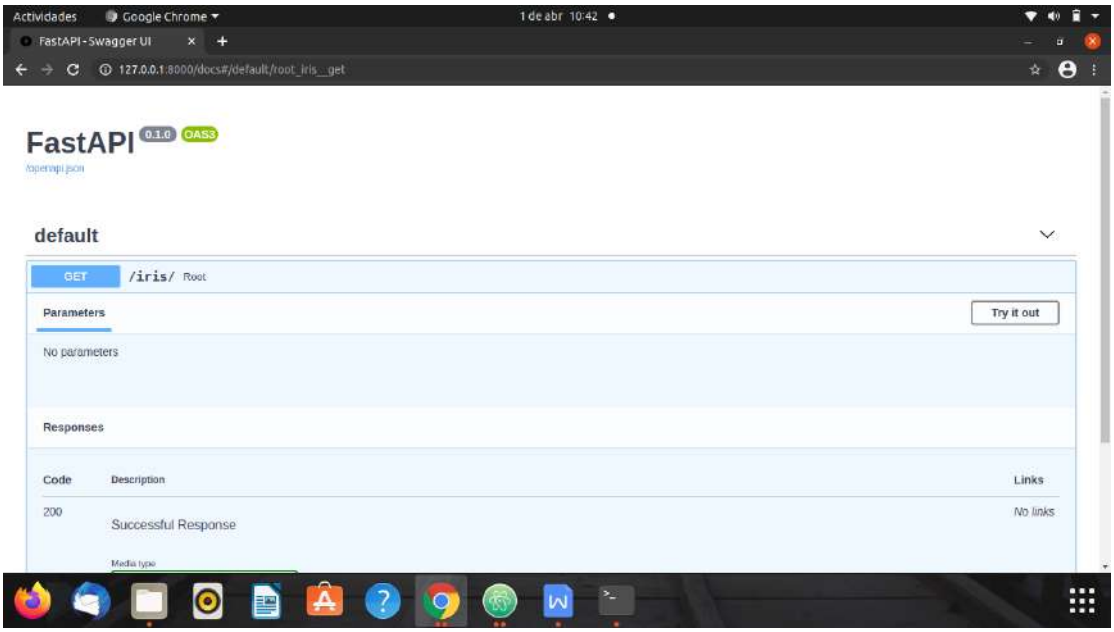

Figura 3.4. Ejemplo de programación del método GET en FastAPI con Iris Dataset

Nos muestra habilitado el método GET a la ruta "/iris/" damos a ***Try it out*** --> ***Execute***:

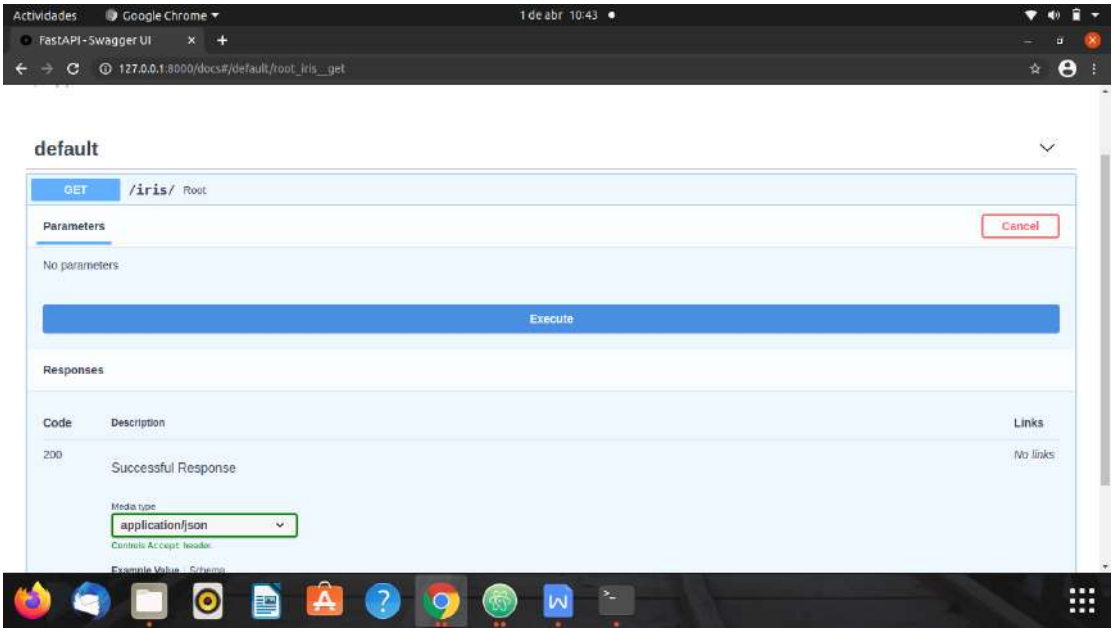

Figura 3.5. Ejemplo de programación del método GET en FastAPI con Iris Dataset

Nos da en ***Responses*** un **200** y un JSON de salida (**Response body**):

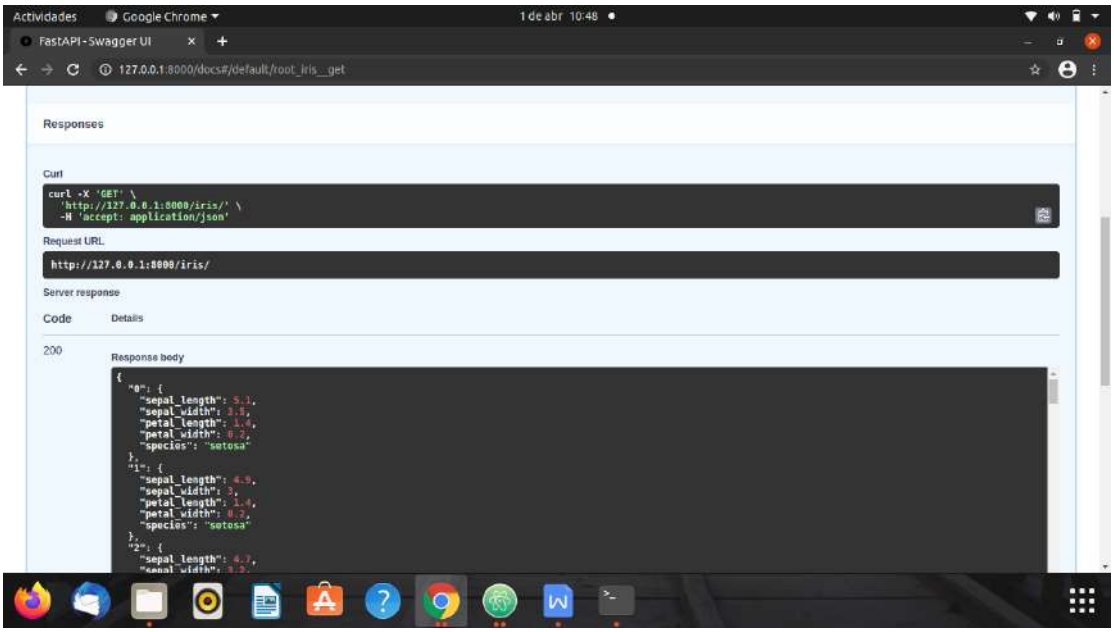

Figura 3.6. Resultado al realizar un GET "/iris/" donde se muestran los datos de Iris dataset

Si ponemos el url: http://127.0.0.1:8000/iris/ en el navegador **Firefox** nos muestra los datos de manera más visual, es también una buena herramienta para navegar por ellos, buscar algún dato, etc.

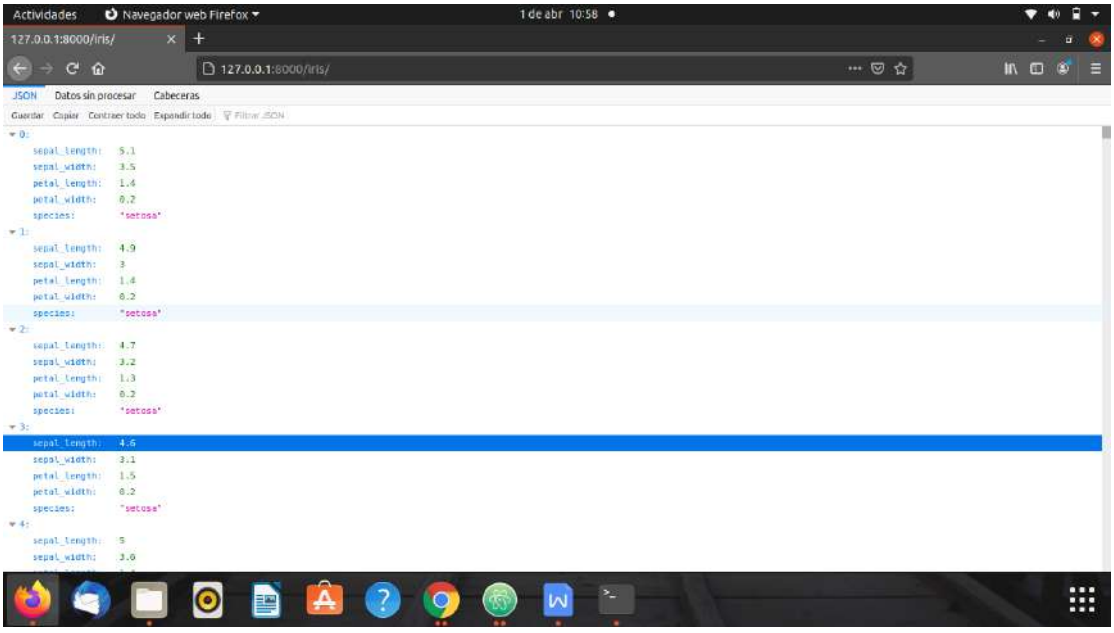

Figura 3.7. Resultado al realizar un GET "/iris/" donde se muestran los datos de Iris dataset

3.2. Método POST

En este método ensayaremos la inserción de un dato nuevo en el CSV de Iris, para ello necesitamos importar la librería:

```
from pydantic import BaseModel --> donde creamos nuestro modelo de datos


Añadimos el modelo de datos:
class Iris(BaseModel):
    sepal_length: float
    sepal_width: float
    petal_length: float
    petal_width: float
    species: str


Creamos la función donde crearemos el dato:
# Método POST a la url "/insertData/"
# llamaremos a nuestra aplicación (<app name> + <método permitido>)
@app.post("/insertData/")
async def insertData(item: Iris):
    # Leemos el archivo iris.csv e
    # insertamos en la última línea los campos a insertar
    with open(MEDIA_ROOT, 'a', newline='') as csvfile:
        # Nombres de los campos:
        fieldnames = ['sepal_length', 'sepal_width', 'petal_length',
                      'petal_width', 'species']
        # escribir en el csv
        writer = csv.DictWriter(csvfile, fieldnames=fieldnames)
        # insertar en la última fila:
        writer.writerow({'sepal_length': item.sepal_length,
                         'sepal_width': item.sepal_width,
                         'petal_length': item.petal_length,
                         'petal_width': item.petal_width,
                         'species': item.species})
    return item
```

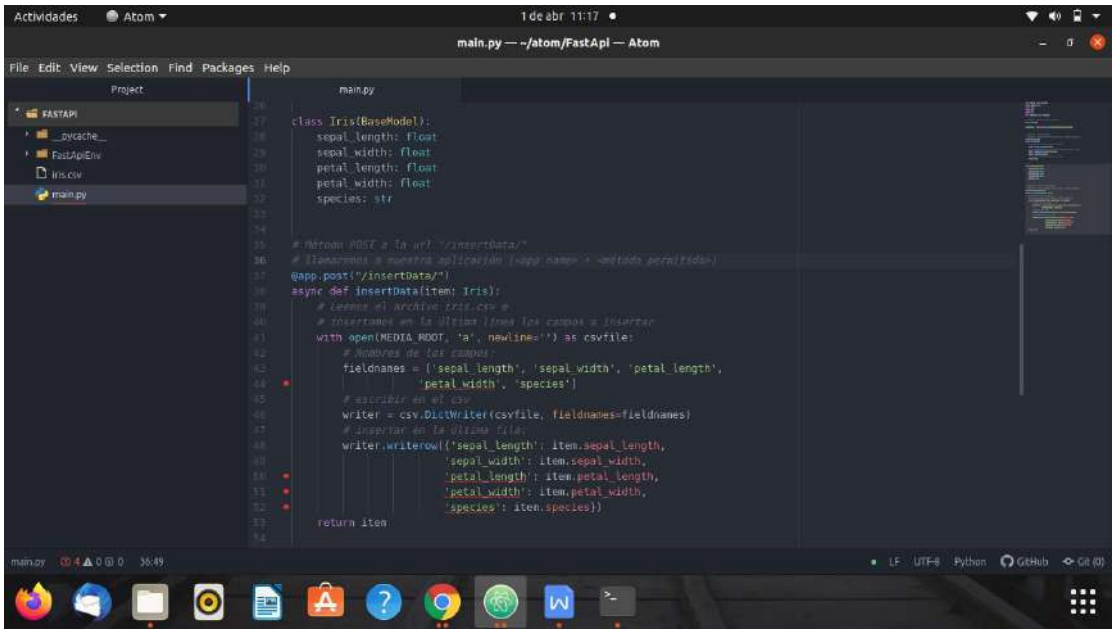

Figura 3.8. Programación del método POST en FastAPI, creando un modelo de datos

Una vez que tenemos ya la función creada nos vamos a la url: http://127.0.0.1:8000/docs

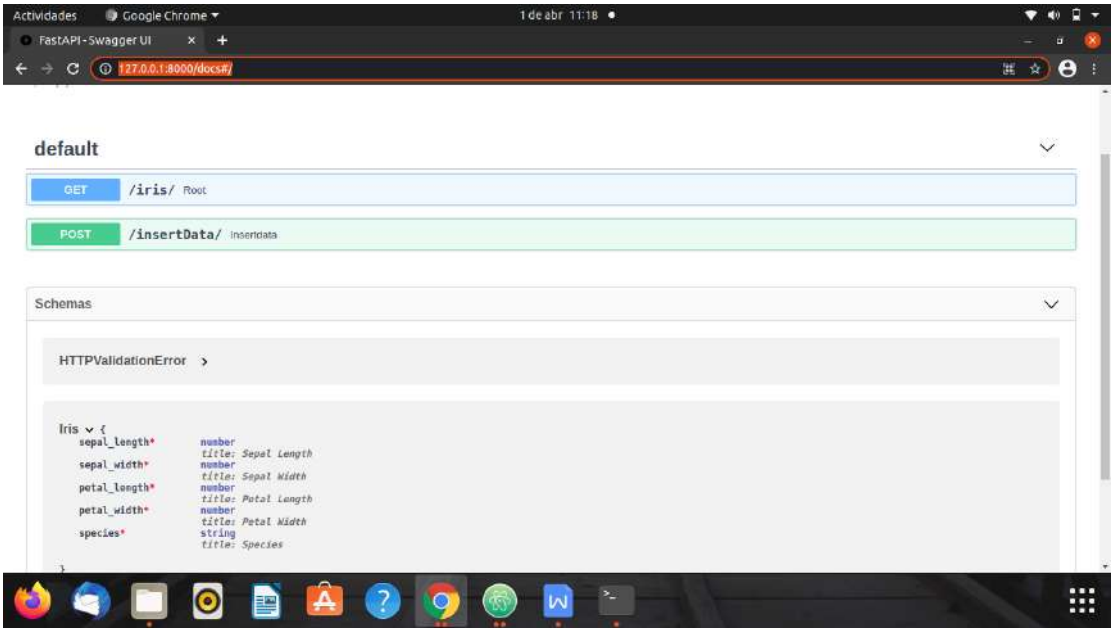

Figura 3.9. Se observa que se ha habilitado el método POST y en la parte de ***Schemas*** el modelo de datos

Observamos que nos habilita para realizar el método POST a la url "/insertData/", además en la parte de ***Schemas*** aparece nuestro modelo de datos que espera el método POST llamado Iris.

Si desplegamos ese método y damos a ***Try it out***:

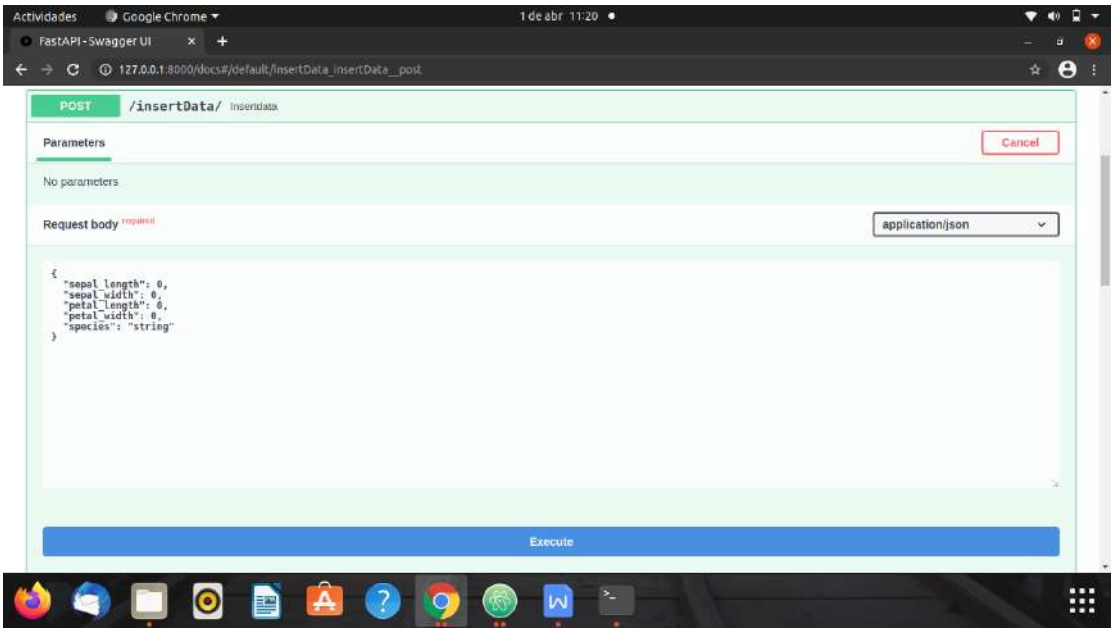

Figura 3.10. Se observa que en ***Parameters*** nos habilita en ***Request body*** los datos que podemos enviar y el formato

Nos aparece un ***Request Body*** con el modelo de datos que tenemos que modificar para insertar en nuestro CSV, modificamos los datos:

{ "sepal_length": 3.4, "sepal_width": 2.3, "petal_length": 1.3, "petal_width": 0.4,

"species": "versicolor" }

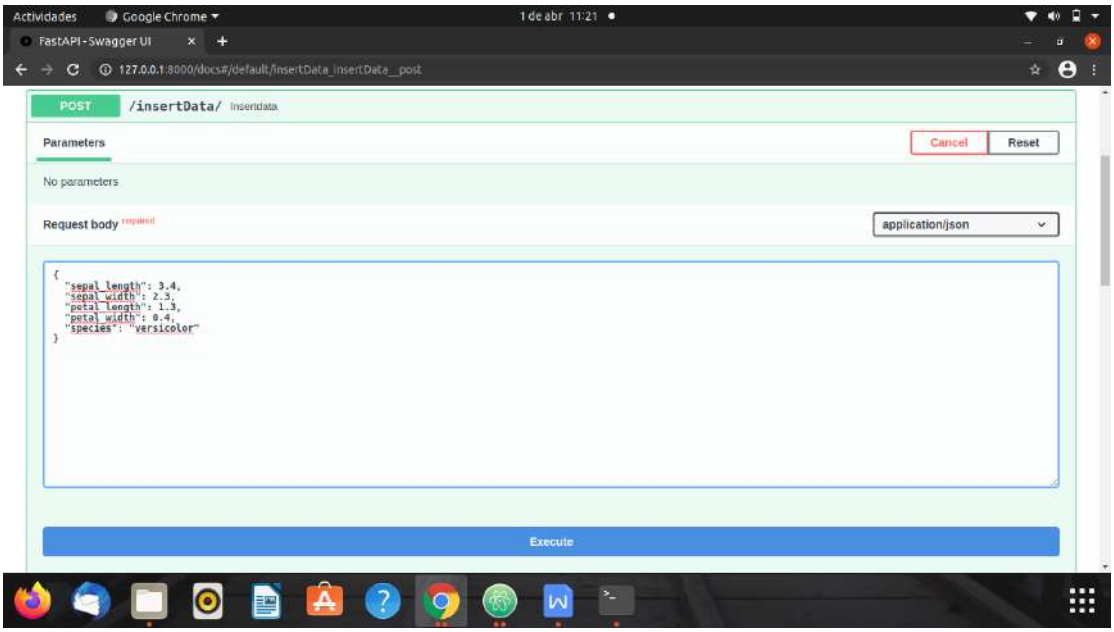

Figura 3.11. Ejemplo de dato a enviar para insertar en el CSV de iris

Pulsamos ***Execute***:

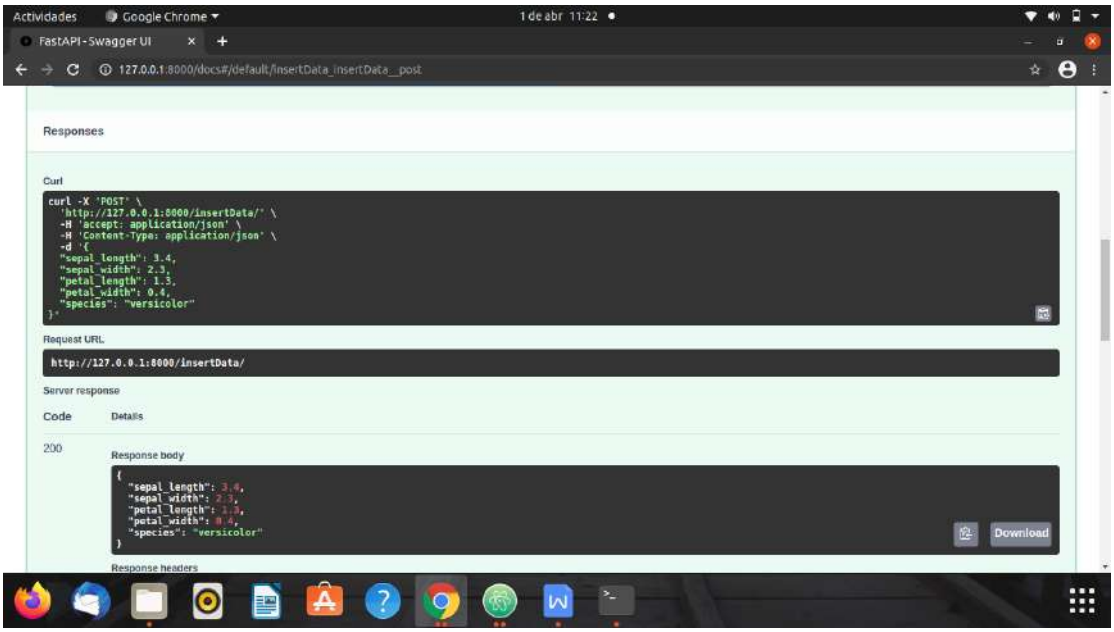

Figura 3.12. Respuesta al POST

En ***Responses*** nos muestra un **200** y el dato que se ha insertado.

Si vamos a nuestro csv:

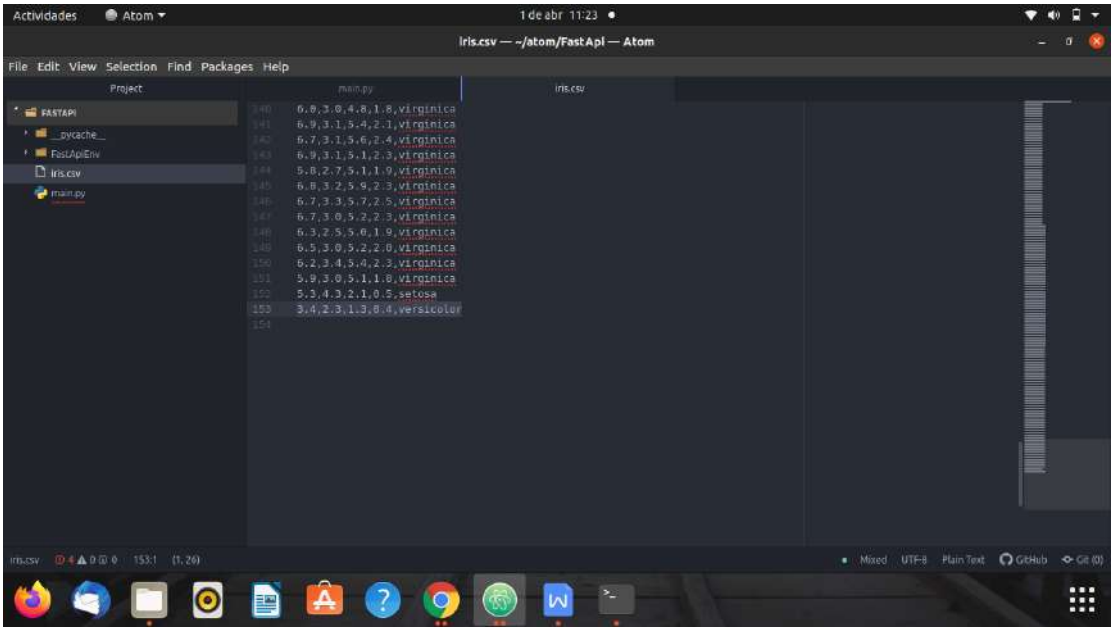

Figura 3.13. En la imagen se observa cómo se ha insertado el dato en el CSV de Iris

Observamos que se ha insertado ese dato.

3.3. Método PUT

En este método PUT realizaremos la actualización del último dato del CSV de Iris dataset.

Para el método PUT crearemos la función:

```
# Método PUT a la url "/updateData/" + el ID del último dato (150)
# llamaremos a nuestra aplicación (<app name> + <método permitido>)
@app.put("/updateData/{item_id}")
async def updateData(item_id: int, item: Iris):
    # Leemos el csv con ayuda de pandas:
    df = pd.read_csv(MEDIA_ROOT)
    # Modificamos el último dato con los valores que nos lleguen:
    df.loc[df.index[-1], 'sepal_length'] = item.sepal_length
    df.loc[df.index[-1], 'sepal_width'] = item.sepal_width,
    df.loc[df.index[-1], 'petal_length'] = item.petal_length,
    df.loc[df.index[-1], 'petal_width'] = item.petal_width,
    df.loc[df.index[-1], 'species'] = item.species
    # convertir a csv
    df.to_csv(MEDIA_ROOT, index=False)
    # Retornamos el id que hemos modificado y el dato en formato diccionario:
    return {"item_id": item_id, **item.dict()}
```

Vamos a la url: http://127.0.0.1:8000/docs#/

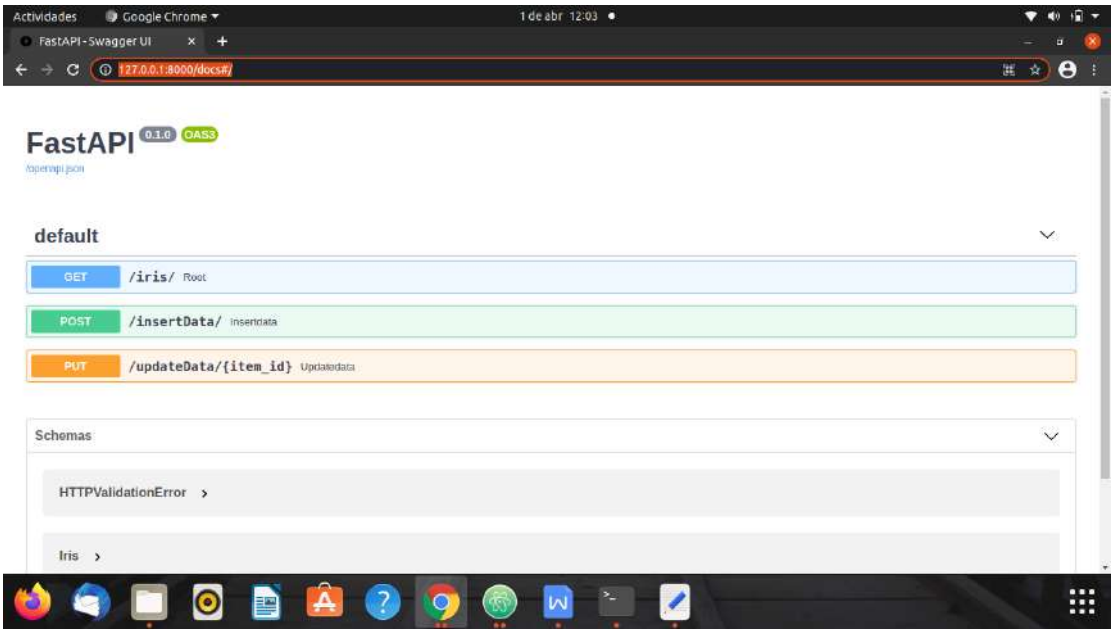

Figura 3.14. En la imagen se observa que se nos ha habilitado el método PUT

Para mandar el dato a la API pulsaremos en PUT en ***Try it out***:

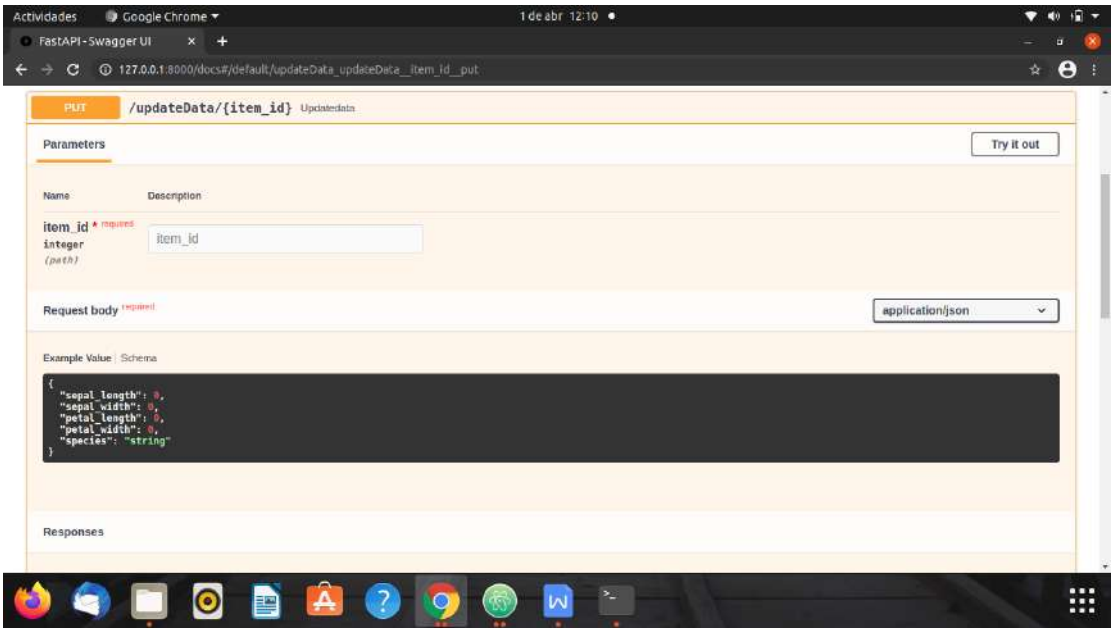

Figura 3.15. En la imagen se observa que se nos ha habilitado el método PUT

En ***Parameters*** pondremos el ID en nuestro caso es el 150 el que queremos modificar y modificamos el ***Request body***.

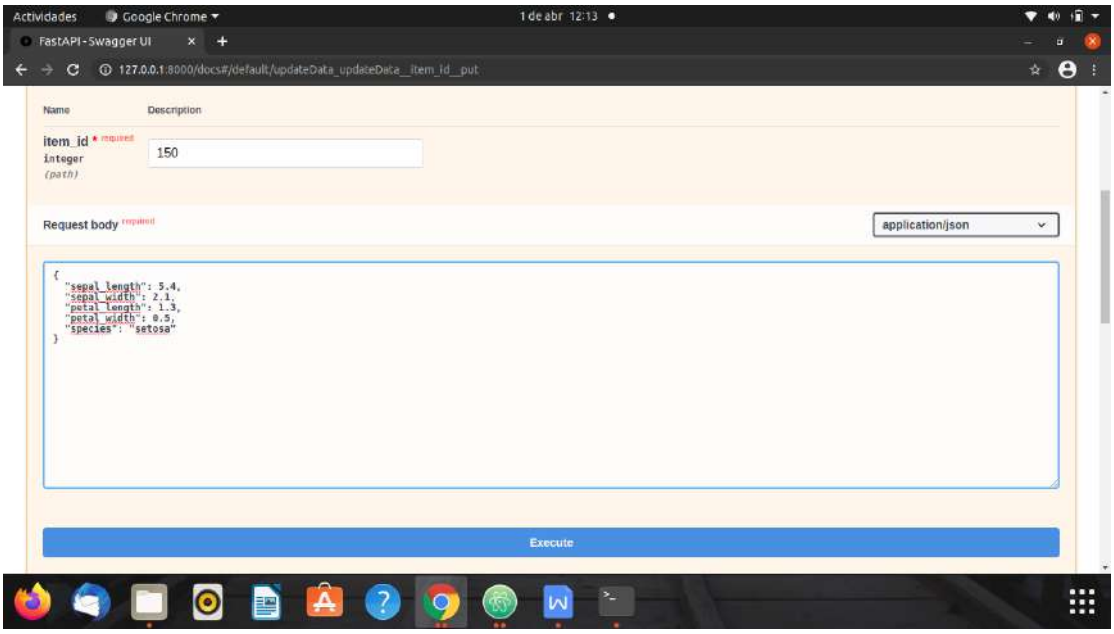

Figura 3.16 Ejemplo de cómo actualizar un dato empleando el método PUT

{ "sepal_length": 5.4, "sepal_width": 2.1, "petal_length": 1.3, "petal_width": 0.5,

"species": "setosa"}

Damos a ***Execute***:

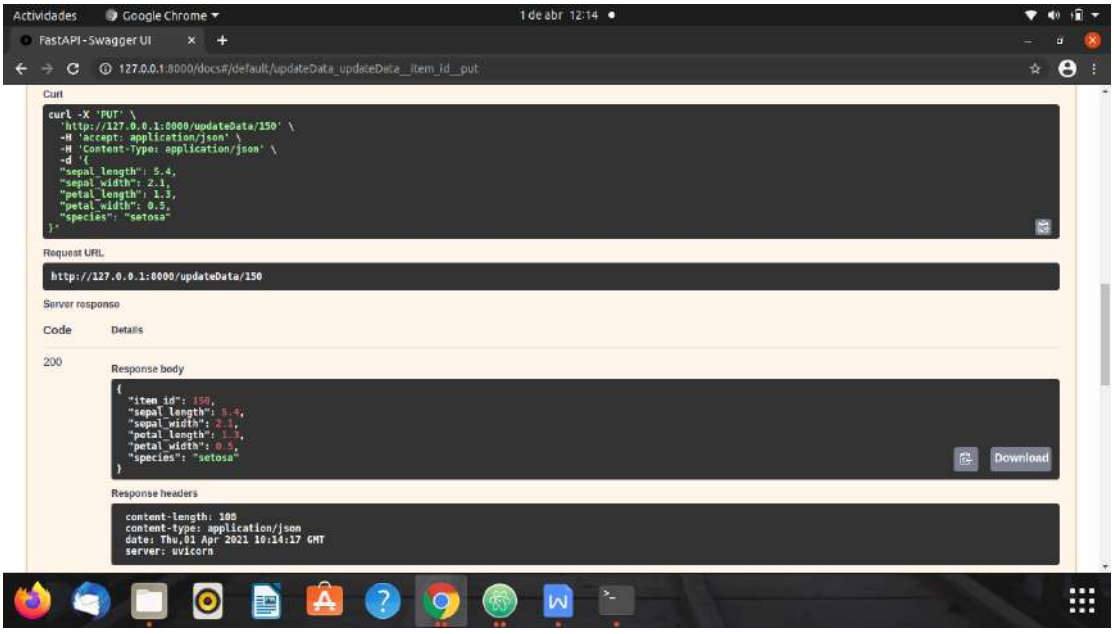

Figura 3.17. Ejemplo de cómo actualizar un dato empleando el método PUT

Nos muestra un **200** y el resultado es un JSON con el item_id más el resto de campos enviados.

Para comprobar que se ha actualizado correctamente vamos al CSV de Iris y vemos si se ha modificado:

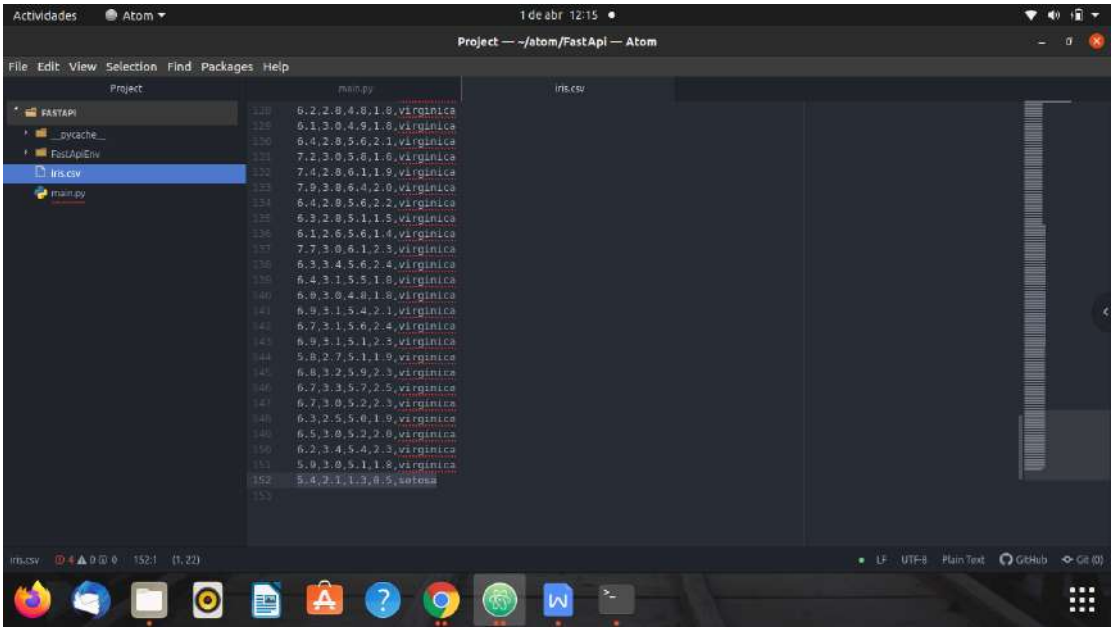

Figura 3.18. En la imagen se observa cómo se ha actualizado el último dato del dataset

Observamos que se ha modificado correctamente.

3.4. Método DELETE

Para eliminar un dato usaremos el método DELETE, definimos la función delete:

```
# Método DELETE a la url "/deleteData/"
# llamaremos a nuestra aplicación (<app name> + <método permitido>)
@app.delete("/deleteData/")
async def deleteData():
    # Leemos el csv con ayuda de pandas:
    df = pd.read_csv(MEDIA_ROOT)
    # Eliminar la última fila
    df.drop(df.index[-1], inplace=True)
    # convertir a csv
    df.to_csv(MEDIA_ROOT, index=False)
    return 'Eliminado'
```

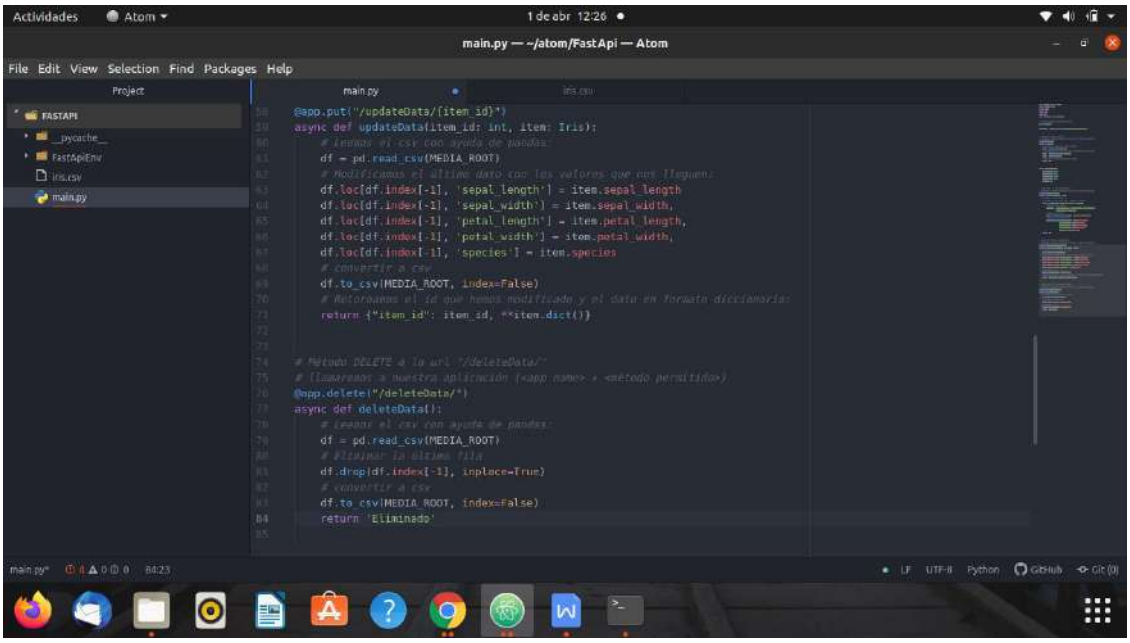

Figura 3.19. En la imagen se observa cómo programar el método DELETE en FastAPI

Vamos a nuestra API en la url: http://127.0.0.1:8000/docs#/

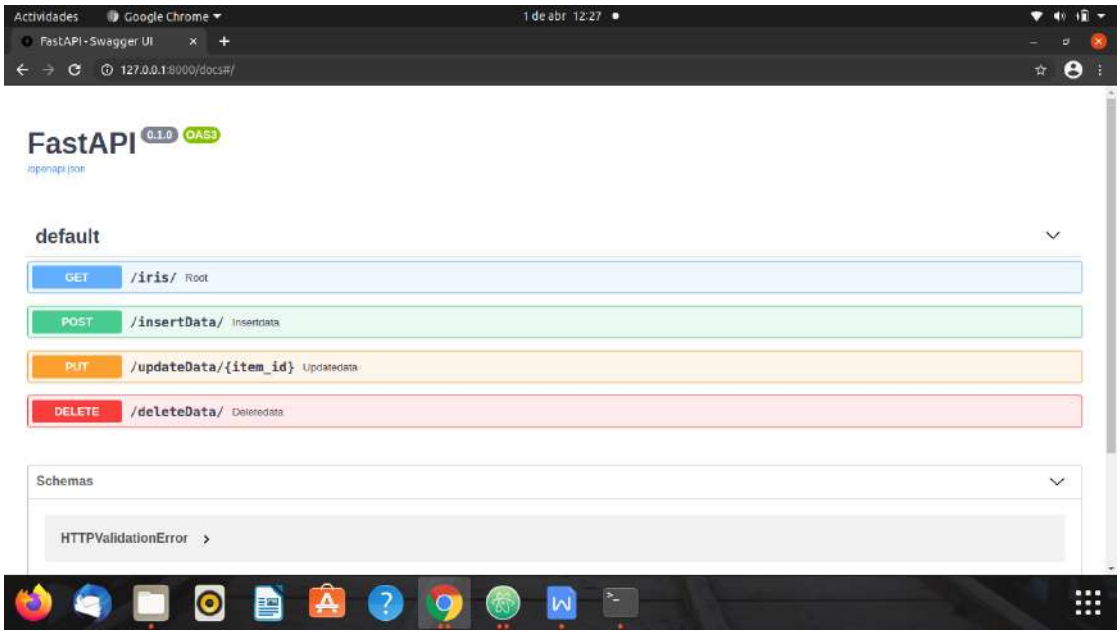

Figura 3.20. En la imagen se observa cómo se ha habilitado el método DELETE en FastAPI

Hacemos clic en el método DELETE:

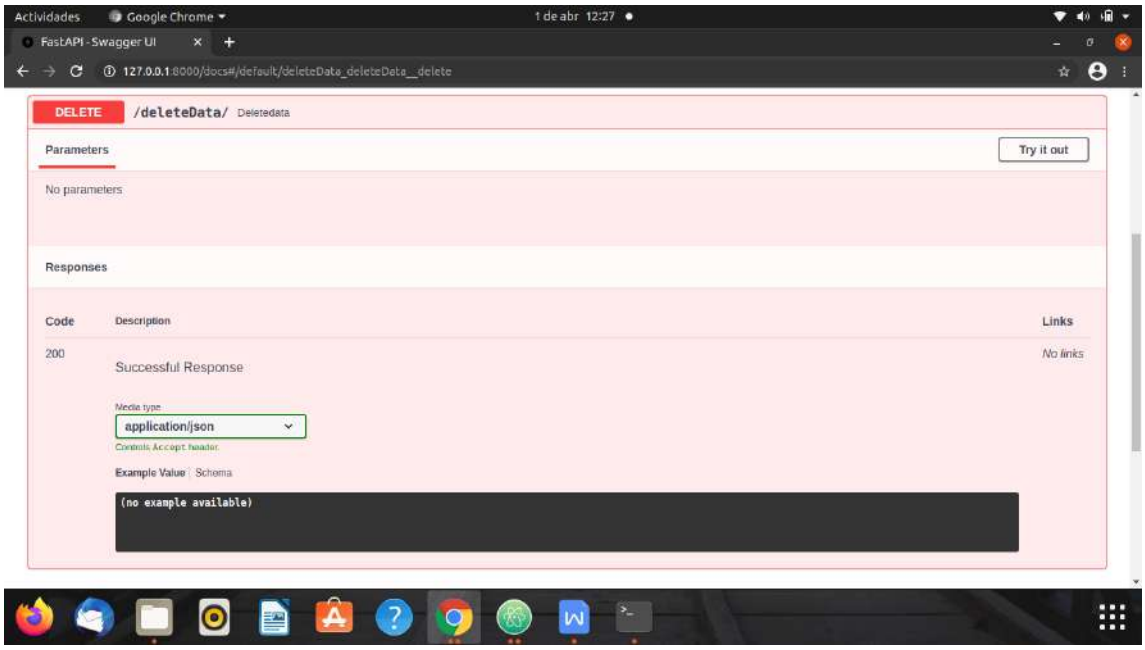

Figura 3.21. En la imagen se observa cómo se ha habilitado el método DELETE en FastAPI

Pulsamos ***Try it out***:

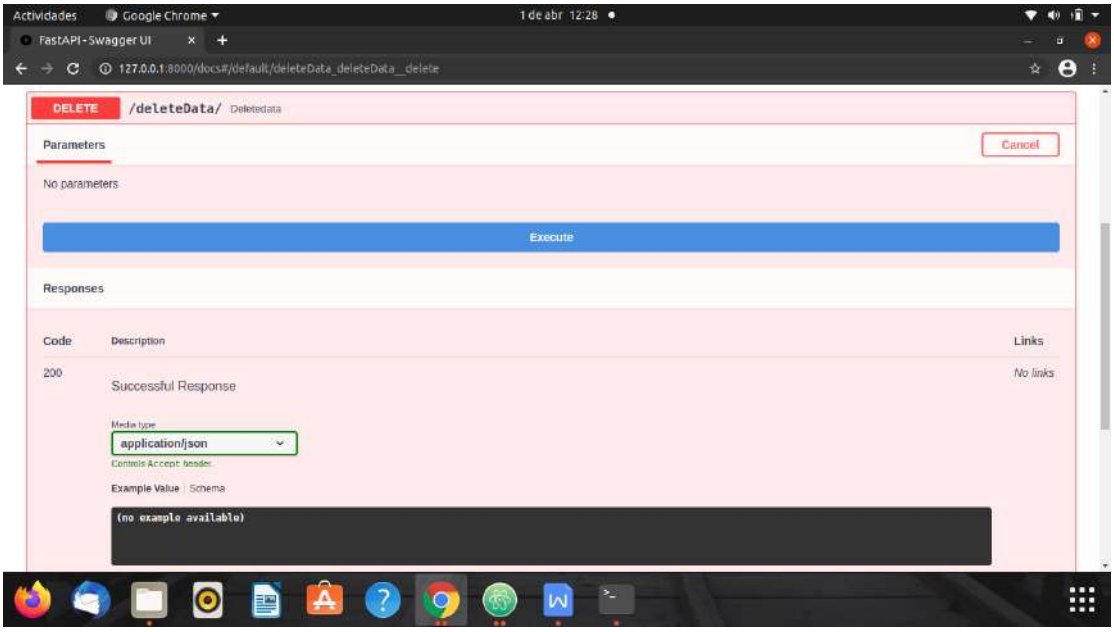

Figura 3.22. En la imagen se observa que al pulsar Try it out no muestra un botón de Execute

Pulsamos Execute:

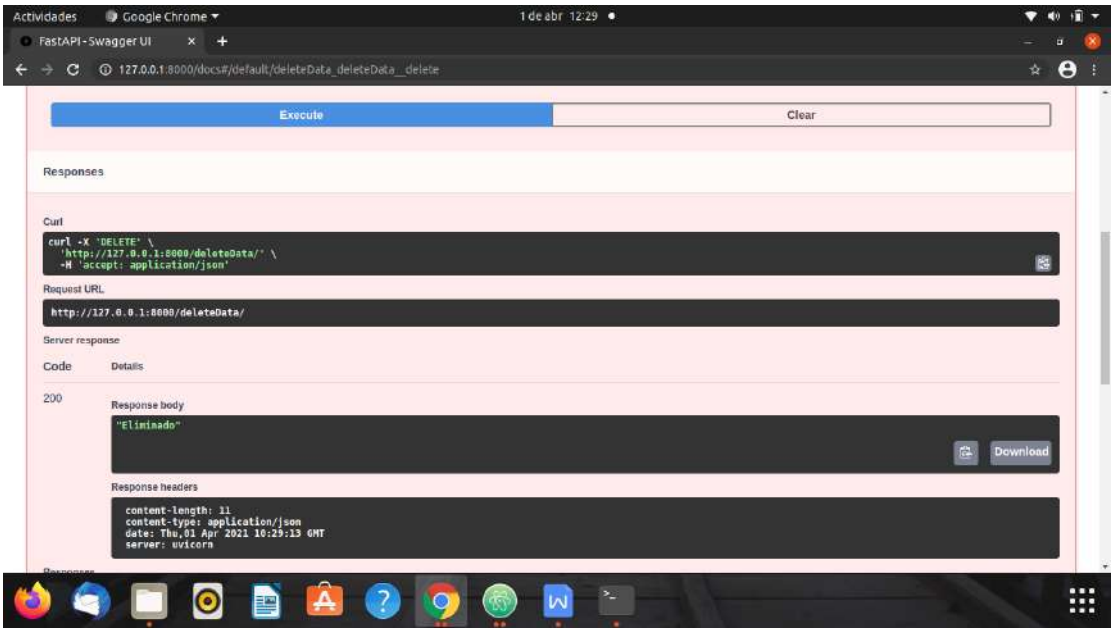

Figura 3.23. En la imagen se observa que ocurre al pulsar el botón Execute

Respuesta es un **200** y el ***Response body***: Eliminado.

4. PUNTOS CLAVE

- Crea un entorno virtual siempre que empieces un proyecto.
- En este caso hemos definido nuestros métodos en el archivo main.py
- FastAPI contiene una URI "/docs" que nos sirve para comprobar la funcionalidad de nuestra API REST.